Wireless Sensor Network, **2010, 2, 891-904**
doi:10.4236/wsn.2010.212107 Published Online December 2010 (http://www.scirp.org/journal/wsn)

Message Cab (MCab): Partition Restoration in MANETs Using Flexible Helping Hosts

Ting Wang, Chor Ping Low
School of Electrical and Electronic Engineering,Nanyang Technological University, Singapore
Email: *wang0235@e.ntu.edu.sg*, *icplow@e.ntu.edu.sg*
Received September 18, 2010; *revised September* 26, 2010; *accepted October* 12, 2010

Abstract

Helping hosts are intensively used in various schemes to restore partitioned Mobile Ad Hoc Networks (MANETs). Most of the existing schemes offers only deterministic deployment and fixed routes for the helping hosts, and are thus not able to deal with fluctuating network traffic, which is a practical condition in many MANET applications. In this paper, we argue that flexible helping hosts (referred to as Message Cabs (MCabs)), with deployment and routes that response to the changes in the traffic demand of the network, may overcome this drawback and reduce the message delay in the networks. To demonstrate the effectiveness of this observation, we propose a new helping host scheme namely the Message Cab (MCab) scheme for partition restoration in MANETs, and validate the performance through simulations.

Keywords: Mobile Ad Hoc Networks (MANETs), Helping Host, Message Cab, Adaptive Route Design, Dynamic Deployment

1. Introduction

A Mobile Ad Hoc Network (MANET), as described in [1], is a kind of mobile wireless networks which is comprised of a collection of mobile hosts connected through wireless channels. The direct connections between the hosts are referred to as links. A host in a MANET exchanges messages with other nodes as a terminal and forwards the messages as an intermediate router at the same time. The routing paths of a message in a MANET are formed by a series of mobile hosts. Messages are forwarded hop-by-hop.

While the mobility of hosts enables the network to span over a large area, it also causes a highly dynamic topology, which is a major challenge to the applications of MANETs. When a host moves out of another's communication range, the link between them breaks, and the entire message routing path may be destroyed by this broken link. This possibility of link breakage may split hosts into different parts between which there is no possible path. This in turn may result in packets not being able to reach their destinations. We refer to this phenomenon as network partitioning. Each isolated part of a network is referred to as a partition of the network. The partitioning problem makes a critical strike on ad hoc routing because most protocols typically assume that the network is always connected. To enhance the reliability and conserve energy, partitioning should be restored in MANETs.

Various approaches have been proposed for MANETs survivability and restoration. In particular, helping hosts are deployed to reconnect the network partitions, and we refer to such action as partition restoration. The helping hosts are able to reconnect the network connectivity by moving from one partition to another in a MANET despite the fact that the normal hosts (i.e. nonhelping hosts) may be partitioned. This idea is also extensively discussed in Delay Tolerant Networks (DTN) and Wireless Sensor Networks (WSN) in order to collect data from disconnected parts of the networks.

The helping hosts are addressed by different names in various schemes. In [2-7], they are referred to as ferries, while in [8-10], they are called helping nodes or forwarding nodes. Data MULEs in [11-13] are also known as a kind of helping hosts, and in the DakNet project [14] buses are used as helping nodes to connect broadband network to rural villages.

One of the most important and commonly discussed objectives of these schemes is to enhance the connectivity and minimize the communication delay in the network. On contrary, helping hosts' movement consumes considerable amount of energy and time, and may cause long message delay in the network. To efficiently utilize them, the deployment and route design of helping hosts are crucial.

In this paper, we propose a new helping host scheme for partition restoration in MANETs, namely the Message Cab (MCab) scheme. The helping hosts are like cabs, in the sense that they are more flexible and adaptive to traffic than buses or ferries. The MCab scheme consists of two essential parts, namely the Dynamic Cab Deployment (DCD) algorithm and the Adaptive Cab Route (ACR) algorithm for the purposes of selecting message cabs and improving the message cab route, respectively. Both algorithms are able to deal with fluctuating traffic in the network, which is a practical scenario in real life, but is not usually discussed in the existing works. Therefore we say that the message cabs are flexible helping hosts. We will demonstrate that the MCab scheme overcomes the drawbacks of existing schemes by incurring lower average message delay and the results are validated through simulations.

In the following parts of this paper, we introduce the backgrounds of our work in Section 2, and the model together with our objectives in Section 3. The two parts of the Message Cab (MCab) scheme, namely the Dynamic Cab Deployment (DCD) algorithm and the Adaptive Cab Route (ACR) algorithm are presented in Section 4 and 5, respectively. Simulation and results are discussed in Section 6, while Section 7 concludes this paper.

2. Background

Before we present our proposed Message Cab (MCab) scheme, we need to explain two important concepts, namely the deployment and route design of the helping hosts (or cabs in our scheme), together with some existing solutions. In addition, our assumptions and notations are discussed in this section.

2.1. Deployment

Deployment is a procedure for selecting helping hosts. In most of the existing works, such as [2,10,13,14], the helping hosts are a group of specially designed hosts that have larger storage, more energy and/or higher moving speed. They are different from the normal hosts and are assigned as helping hosts prior to the commencement of network operations. The deployment is thus static, or deterministic. Such deployment mechanisms allows helping hosts to have higher capability in delivering messages across partitions but is less flexible. Since the number of helping hosts is fixed, it may turns out that there is too many, or too few helping hosts to meet the traffic demand in the network. Thus static deployment may not be adaptive to network traffic demand. To overcome this drawback, dynamic deployment is used in [8], where normal hosts with suitable moving direction and speed are chosen as helping hosts (a.k.a helping nodes). However, it is a centralized scheme and thus not scalable with network size. More importantly, under the assumption that the network is partitioned, it is infeasible to make all the hosts' movement information available to a central server to perform the selection. Similar problem can also be observed in [9]. Therefore, we propose a localized algorithm, namely the Dynamic Cab Deployment (DCD) algorithm to deploy helping hosts (i.e. message cabs) in the MCab scheme. The DCD algorithm allows the number of helping hosts to change dynamically with the traffic volume, and thereby reduces the weighted average delay of messages in the network, and enhances the scalability of the deployment process.

2.2. Route Design

The route of a helping host is the path it follows, which usually connects different partitions or hosts in the network. In [8,12,14], the route is not planned by any algorithm. The hosts' random movement is utilized to deliver messages. However, it has been shown in many other works that the message delay can be significantly reduced if we consider the problem of route design as an optimization problem. For example, in [2], Zhao *et al.* have formulated the Massage Ferry Route (MFR) problem to find the optimal route for helping hosts (i.e. message ferries). In [4-7], solutions from the well studied Traveling Salesman Problem (TSP) and its variants are adopted as routes for helpinghosts. In these schemes, the route does not change after being constructed, and is thus a fixed route. It is only suitable when the network traffic among the partitions is constant, which is an idealized assumption. In most of the practical cases, the traffic between any two partitions will be a temporal variable, and the route should be able to adapt to the changes. Hence fixed routes are unlikely to be optimal in general. Therefore, an adaptive route is more desirable. Ou *et al.* proposed a centralized scheme in [10] to let the helping hosts (a.k.a helping nodes) move adaptively to the packets' destinations. However, as we have discussed in the previous section, centralized schemes may perform poorly when the network is partitioned. In this paper we propose a distributed algorithm, namely the Adaptive Cab Route (ACR) algorithm, to construct cab routes by adopting the Shortest Process Time First (SPTF) rule from the Job Sequencing Problem (JSP). Simulation proves that our proposed scheme is able to effectively reduce the message delay in the network.

2.3. Assumptions and Notations

While many existing schemes (e.g. [2,7,11]) focus on stationary hosts with known locations and predictable network traffic, we try to deal with a more practical scenario, where:

- the hosts are mobile;
- the traffic in the network is randomly variant and is thus not predictable.

As discussed in [15], movements of hosts in MANETs can be considered as Levy Flights, in which the hosts tend to stay in a certain area for a long time and occasionally make long distance flights to places far away. We could thus assume that a host stays in the same cluster for a certain period of time and will also sometimes decides to move to another cluster. Practically, this could be due to the assignment of tasks to the host, such as in a wireless sensor network, a sensor may need to reallocate itself to new positions to collect new data.

Let's assume that the MANET of our interest has n_h mobile hosts. We assume the network is partitioned to n_c components, each of which forms a cluster with a chosen cluster head. The hosts in the same cluster are always connected. In order to simplify the problem, we restrict the movement of a cluster head to be inside a circular area, which is referred to as the head zone. The radius of head zone is equal to the communication range of the cluster head, denoted as *r*, as shown in **Figure 1**. The center of head zone of cluster s is denoted as point $1 \le s \le n_c$. We use ε_{sd} to denote the time taken by a host to travel from C_s to $C_d, 1 \le s, d \le n_c$. We note that $\in_{sd}$ is a positive real number. We say cluster d is a neighbor of cluster s if a host is able to move directly to cluster d from cluster s in one hop.

Using proper clustering schemes, such as [16], the cluster head will always be aware of the changes in its cluster members, such as the movements, arrivals of new members and departures of existing members. It also knows the locations of other clusters, so when a host decides to move to a new place, the cluster head knows which is the next cluster the host is going to join. In addition, the cluster head works as a proxy between other normal hosts in the cluster and the helping host as well. If a helping host stops at C_s, it is able to communicate with the head of cluster s and deliver the messages to other hosts in cluster s via the cluster head. The route of a helping host, denoted as R, will be a sequence of head zone centers (C_s's). Each of these head zone centers is like a waypoint (as illustrated in **Figure 1**), which is a temporary destination for the helping host. Upon arriving a waypoint, the helping host stays for a period of time to deliver and receive the messages to/from the cluster head, before moving on towards the next waypoint. The route segment between two consecutive waypoints is referred to as a leg (shown in **Figure 1**).

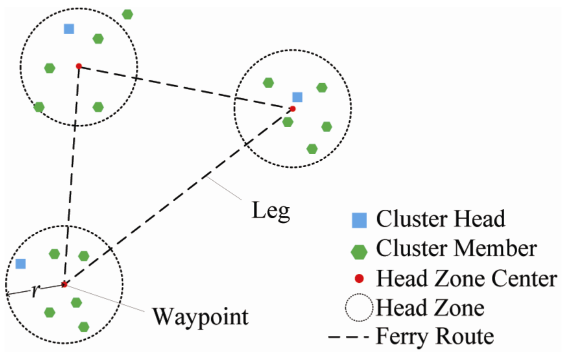


Figure 1. Network topology.

3. System Model and Objectives

To demonstrate our methodology of the MCab scheme, we describe an analogy between the MANET and a simple transportation system which serves several small towns. It will also explain the reason why we name our helping hosts as cabs and how they are different from ferries.

The towns represent the clusters, each of which is distant away from the others. The cars are like the hosts—they usually move within a town, and when it is necessary, they are also able to move from one town to another. The passengers are akin to messages, which by themselves cannot move from town to town.

Buses are provided as an existing solution to the inter-town transportation problem. They are similar to the other statically deployed helping hosts (such as message ferries) that we have discussed in the previous section. The static deployment and fixed route design of buses perform poorly with varying volume of passengers. Hence, it would be preferable if we have a flexible solution.

Besides taking buses, the passengers could consider hiring a cab to travel to another town. The route of a cab will depend on the passengers' demands, and is thus more flexible than the bus routes. Practically, it is also more convenient in terms of saving time to hire a cab. The number of cabs is determined by how many passengers there are. When there are more passengers, more cabs can be recruited from the private cars (normal hosts) available; when the number of passengers drops, some cabs could retire and become private cars again.

Moreover, since it is possible that some cars will move to another town by their drivers' own decisions, a passenger may also take a ride from a private car which is also moving to the his/her destination town. This will utilize the mobility of private cars in the transportation system to serve the passengers' needs and save their traveling time. A car offering ride to passengers is like a one-time temporary cab that only works for a single trip to a particular town. We refer to such cars as temp-cabs. To distinguish from them, we refer to those cabs that works for multiple trips in a longer period of time as professional cabs, or pro-cabs.

We imitate the method of hiring pro-cabs and taking rides from temp-cabs in the MANETs and propose the Message Cab (MCab) scheme. As shown in **Figure 2**, the state of a private car (normal host), which is not a cab nor a cluster head, can transform to a pro-cab or a temp-cab under the control of the Dynamic Cab Deployment (DCD) algorithm:

Pro-Cab

Its movement is passively determined by the Adaptive Cab Route (ACR) algorithm, in which the route is adaptively designed based on the network traffic. It is recruited from a normal host, and continues moving among the clusters for a certain predetermined period of time before retiring to be a normal host.

Temp-Cab

Its movement is pro-actively determined by the host itself. It is a one-time helping host that only leaves its original cluster to its destination cluster. It becomes a normal host again after it arrives and has delivered the inter-cluster messages to the destination cluster.

The key difference between message cabs and the other types of helping hosts in the existing works (such as ferries, helping nodes etc.) is that they are not selected before the network starts. Therefore the number of cabs and cab routes can dynamically change with the traffic. Although this kind of deployment does not allow the helping hosts to have higher capability in moving speed or storage space (since message cabs are just selected normal hosts), we found that it can still effectively reduce the message delay in the network.

Since hosts in the same cluster are connected, we can use existing MANET routing protocols such as AODV or DSR for the communication between a cluster head and its members. We note that the delay incurred by the message transmission within a cluster (intra-cluster communication) is much smaller than the delivery between different clusters (inter-cluster communication), and is less relevant to the cab deployment and route design. As a consequence, intra-cluster communication will be omitted in our following discussion.

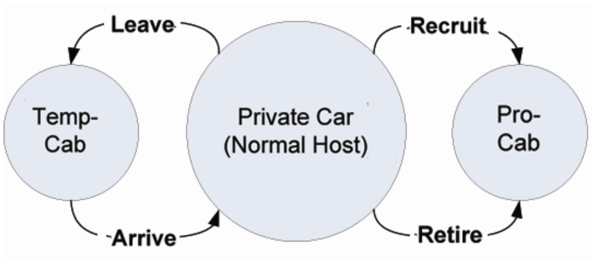


Figure 2. States of a host

We should note that before a message is collected by a cab, it needs to wait at some cluster head for a certain time duration. We refer to this amount of time spent on waiting as the waiting delay (ω) of the message. After it is collected by a cab, the cab will travel from its source cluster to its destination cluster, and thus incurs a traveling delay (τ) for the message. Therefore, the overall delay δ of a message is $\delta = \omega + \tau$.

Assume within time duration (0, t], there are n_m messages that have been transmitted by the message cabs. The size of message i $(1 \le i \le n_m)$ is μ_i. The waiting, traveling and overall delay of message i are denoted as and ω_i, T_i and δ_i .respectively. Similar to the objectives in [2-4], we are interested in reducing the weighted average overall delay (Δ) of the messages:

$$\Delta = \frac{\sum_{i=1}^{n_m} \mu_i \delta_i}{\sum_{i=1}^{n_m} \mu_i} = \frac{\sum_{i=1}^{n_m} \mu_i (\omega_i + T_i)}{\sum_{i=1}^{n_m} \mu_i} = \Delta_\omega + \Delta_T \tag{1}$$

where the weighted average waiting delay (Δ_ω) is given by

$$\Delta_\omega = \frac{\sum_{i=1}^{n_m} \mu_i \omega_i}{\sum_{i=1}^{n_m} \mu_i}, \tag{2}$$

and the weighted average traveling delay (Δ_T) of all the messages can be taken as

$$\Delta_T = \frac{\sum_{i=1}^{n_m} \mu_i T_i}{\sum_{i=1}^{n_m} \mu_i}, \tag{3}$$

We can see that if there are more cabs in the network, each cluster could be visited more frequently, and the waiting delay of messages reduces. It shows that the waiting delay (Δ_ω) is closely related to the cab deployment plan. On the other hand, optimization of cab routes results in a shorter traveling delay Δ_T.

In the MCab scheme, we bound Δ_ω from above by using the Dynamic Cab Deployment (DCD) algorithm to control the number of cabs, and Δ_T is reduced by adopting optimization techniques in the Adaptive Cab Route (ACR) algorithm, which designs the routes of the cabs.

4. Deployment of Message Cabs

In the ideal scenario, once a inter-cluster message is received by the cluster head, there is always a cab ready to carry it towards its destination, and Δ_ω will be 0. However, in practice, with unpredictable and fluctuating traffic, it is impossible to have a cab ready for message delivery as soon as a message arrives at the cluster head. It would also be inefficient if a cab carries only one message and does not fully utilize its storage space. To solve this problem, we try to make use of the normal hosts mobility as a temp-cab to give a ride to the messages as well as to recruit pro-cabs from normal hosts in the DCD algorithm. We show that by doing so, we are able to bound the weighted waiting delay of messages from above by a predetermined value, denoted as Ω seconds.

Since the change of cluster membership is handled by the cluster head, a host (say host a) needs to report to the head of its current cluster, say cluster s (as source) before it leaves for a new location. The cluster head of clusters then knows the new cluster (say cluster d, as destination) host a is going to join. It becomes possible that the head of cluster s lets host a to carry some messages which have cluster d as their destinations. Host a a will then deliver these messages to the cluster head d when it arrives there and registers itself as a new cluster member. Therefore, when a host is leaving a cluster, its state changes from normal host to temp-cab, and it is used to deliver as many messages as possible across the partitioned clusters. Upon its arrival, it delivers the messages to the head of the new cluster and its state changes back to that of a normal host. This procedure of deploying a temp-cab is depicted in **Figure 3(a)**.

On the other hand, a timer is used to control the time when a pro-cab should be selected. Assuming there are n'_m inter-cluster messages stored in cluster head s, of each the size is μ_i. The current waiting delay of a message is the length of the time duration since the message is received by the cluster head, and is denoted as ω'_i. The timer is set to

$$t = \Omega - \frac{\sum_{i=1}^{n'_m} \mu_i \omega'_i}{\sum_{i=1}^{n'_m} \mu_i},$$

so that if a pro-cab is recruited right before the timer expires, the average weighted delay of the messages stored in cluster head s will be less than Ω, which is the required upper bound. We note that the value of t needs to be updated every time there is a change in the number of inter-cluster messages which are stored in cluster head s, such as when a new inter-cluster message is received and stored, or when some messages have been uploaded to a cab (either temp or pro) by cluster head s.

We let the time duration between the recruitment and retirement of a cab be denoted as Θ. After having served for Θ seconds as a pro-cab, it retires. Upon its re tirement, a pro-cab does not stop immediately. It continues moving among the clusters to deliver the remaining messages it has already collected, but without collecting new messages. When all the messages are delivered, it moves back to the cluster where it has been recruited, and register with the head as a normal host again.

Figure 3 depicts the flowcharts for the DCD algorithm. To avoid going into too much details, we make some simplifications to the algorithm:

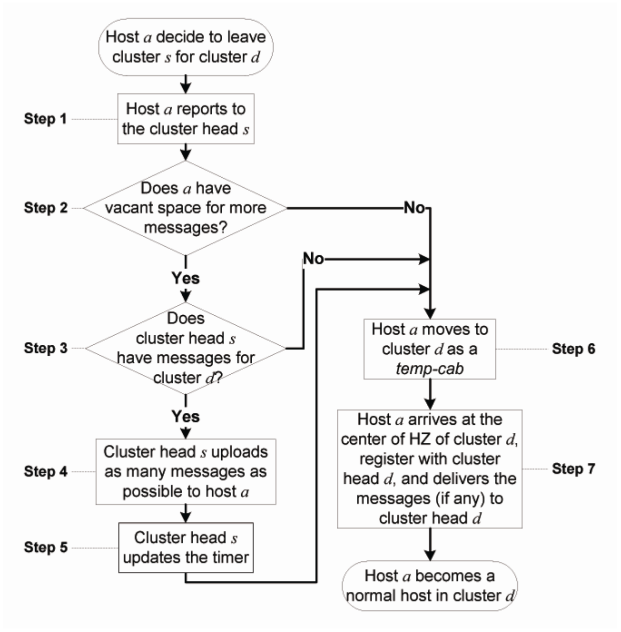
- The pro-cabs are randomly selected from the normal hosts in the cluster;
- If a cab does not have enough space to store all the messages for inter-cluster delivery, it delivers those messages with larger weighted waiting delay $(\mu_i \omega'_i)$ first;
- When the memory of a cluster head overflows, those messages with larger weighted waiting delay $(\mu_i \omega'_i)$ will also be dropped first (*i.e.* the hot-potato rule[1]);
- The value of Θ (*i.e.* the length of time duration between the recruitment and retirement of the cabs) is a constant.

With these simplifications, the DCD algorithm is a fully distributed scheme which does not require any global information.

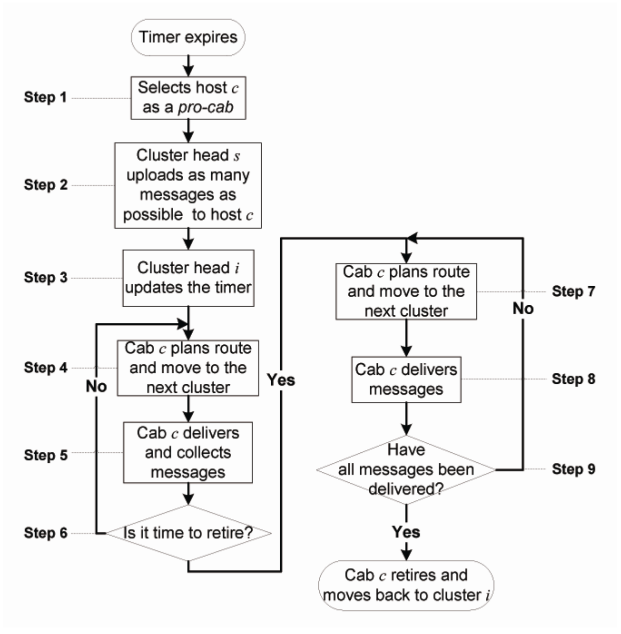
Lemma 1. The worst case complexity of the DCD algorithm is $O(n_m)$, where nm is the number of messages that have been generated during the period of network operation.

Proof: Based on the algorithm depicted in **Figure 3**, we can observe the fact that the complexity of the procedures is independent of the number of clusters and the number of hosts. To update the timer (refer to step 5 in **Figure 3(a)** and step 3 in **Figure 3(b)**), the cluster head needs to go through the messages that have been stored in its memory space (n'_m messages). In the worst case, all the nm messages are stored, and it thus requires $O(n_m)$ time. On the other hand, to upload the messages to the cab (step 4 in **Figure 3(a)** and step 2 in **Figure 3(b)**), and to deliver the messages to another cluster head (step 7 in in **Figure 3(a)**, step 5 and step 8 in **Figure 3(b)**), it requires $O(n_m)$ time as well. We note that the planning of routes (step 6 in in **Figure 3(a)**, step 4 and step 7 in **Figure 3(b)**) will be handled by the ACR algorithm, and is not part of cab deployment process handled by the DCD algorithm; thus its complexity will not be counted here. In addition, step 1 in **Figure 3(a)** is handled by the clustering scheme, and the pro-cab selection (step 1 in **Figure 3(b)**) is random according to our assumptions. Therefore they do not incur additional com-

[1]We deal with the elements that have heaviest impact to the system by either processing them or dropping them first. By dropping those messages with heavier weighted delay, the average weighted delay of the success-fully delivered messages can be reduced.


Host a decide to leave cluster s for cluster d
Step 1
Host a reports to the cluster head s
Step 2
Does a have vacant space for more messages?
No
Yes
No
Step 3
Does cluster head s have messages for cluster d?
Yes
Step 4
Cluster head s uploads as many messages as possible to host a
Step 5
Cluster head s updates the timer
Host a moves to cluster d as a *temp-cab*
Step 6
Host a arrives at the center of HZ of cluster d, register with cluster head d, and delivers the messages (if any) to cluster head d
Step 7
Host a becomes a normal host in cluster d

(a)Temp-Cab


Timer expires
Step 1
Selects host c as a *pro-cab*
Step 2
Cluster head s uploads as many messages as possible to host c
Step 3
Cluster head i updates the timer
Step 4
Cab c plans route and move to the next cluster
No
Yes
Step 5
Cab c delivers and collects messages
Step 6
Is it time to retire?
Cab c plans route and move to the next cluster
Step 7
No
Cab c delivers messages
Step 8
Have all messages been delivered?
Step 9
Yes
Cab c retires and moves back to cluster i

(b) Pro-Cab

Figure 3. Flowcharts of the dynamic cab deployment (DCD) algorithm.

plexity to the DCD algorithm. In addition, the "if" clauses (steps 2, 3 in **Figure 3**(**a**) and step 6, 9 in **Figure 3**(**b**) does not incur additional complexity to the algorithm. As a result, the overall complexity of the DCD algorithm is $O(n_m)$

5. Message CAB Route Design

To improve the route of cabs, we define the Message Cab Route (MCR) problem as a generalization of the existing Massage Ferry Route (MFR) problem, and propose the Adaptive Cab Route (ACR) algorithm as a solution. Since only the pro-cabs' route will be designed by our algorithm, we refer to them as cabs for the sake of convenience in this section.

5.1. The Message Cab Route (MCR) Problem

In [2], Zhao *et al*. have formulated the Massage Ferry Route (MFR) problem to find the optimal route for helping hosts (i.e. message ferries), and its objective is to minimize the average delay of the traffic:

$$\Delta^{\mathrm{R}} = \frac{\sum_{1\le s,d\le n_c} \mathrm{b}_{sd}\delta_{sd}^{R}}{\sum_{1\le s,d\le n_c} \mathrm{b}_{sd}} \tag{4}$$

where bsd is the average traffic per unit time from s to d and δ_{sd}^{R} is the time spent by the ferry to travel from s to d on the static route R. But in our case, b_{sd} could not be determined beforehand because we have assumed that the traffic is unpredictable. Moreover, when R is not fixed (i.e. being adaptive instead), the traveling delay from one cluster to another is not a constant. Therefore the solutions for the MFR problem may not perform well for the problem that we are addressing. Hence we define a generalized version of the MFR problem which is able to adapt to both static and dynamic traffic demand and routes. We refer to this problem as the Message Cab Route (MCR) problem:

Definition 1. (The Message Cab Route (MCR) problem) For a given group of clusters, find the optimal visiting sequence for a cab, so that the weighted average traveling delay Δ_T to deliver n_m messages, which is expressed by

$$\Delta_T = \frac{\sum_{i=1}^{n_m} \mu_i T_i}{\sum_{i=1}^{n_m} \mu_i} \tag{5}$$

where T_i is the traveling delay of message i, can be minimized.

We could make the following observations from the MCR problem:

- **The MCR problem is a generalization of the MFR problem.** In the MFR problem, there are two additional assumptions made as compared to the MCR problem, namely the traffic rate (b_{sd}) between two clusters is constant, and the distance from cluster s to d on the given route $R(\delta_{sd}^{R})$ is also a fixed value. That is, for any message i sent from cluster s to cluster d, its traveling delay will be $T_i = \delta_{sd}^{R}$. Let's say the network operates from time 0 to t. The total size of all the messages that have been generated during this period can be expressed as $\sum_{1\le s,d\le n_c} tb_{sd}$. On the other hand, using our notation, since the number of these messages is n_m, and the size of each message is denoted by μ_i, the total size is thus $\sum_{i=1}^{n_m}\mu_i$. Therefore

$$t\sum_{1\le s,d\le n_c} b_{sd} = \sum_{i=1}^{n_m}\mu_i$$

and thus

$$\sum_{1\le s,d\le n_c} tb_{sd}\delta_{sd}^R = \sum_{i=1}^{n_m}\mu_i T_i$$

Equation (5) becomes equivalent to Equation 4 under the assumptions of the MFR problem. Hence it is easy to see that the MCR problem is a generalized version of the MFR problem.

- **The MCR problem is MAX-SNP-hard, and thus polynomial time approximation schemes are unlikely to exists to this problem**. We consider a simpler special case of the MCR problem. Consider a particular instance where a cab collects $n_c - 1$ messages, each for a distinct destination cluster, from cluster s. Before it finishes delivering these messages, other clusters do not upload any new message to the cab. The objective function, Equation (5) therefore becomes

$$\Delta_T = \frac{\sum_{i=1}^{n_c-1}\mu_i T_i}{\sum_{i=1}^{n_c-1}\mu_i}.$$

This is exactly the objective function of the Weighted Minimum Latency Problem (WMLP) [17], where T_i is referred to as the latency of the tour starting from s to the destination cluster of message i, and μ_i is the weight. In the WMLP, we try to find the tour that visits all the vertices in a graph that minimizes the weighted latency, represented by the above equation. Unfortunately, the WMLP is known to be a MAX-SNP-hard problem [18]. Generally, there is no polynomial time approximation scheme for MAX-SNP-hard problems [19]. Even for a more simplified scenario, where μ_i is constant and the traffic is uniform among the clusters, to minimize $\sum_{i=1}^{n_c-1} T_i$, we still need to solve the Traveling Salesman Problem (TSP), which is also known to be NP-hard. Therefore, it is infeasible to completely generate an optimized cab route.

- **A static route cannot be an optimal solution.** This is because in the network, traffic cannot be constant and uniform all the time. If some cluster i receives messages more frequently than others, it will be a waste of time to visit all the other clusters before returning to i, which is what that will take place with a static route. Moreover, due to the nature of some of the MANET tasks, such as area exploration or surveillance, the frequency and sizes of messages may vary over time. A static route could not adapt itself to such changes.
- **The key to finding a suitable cab route is to determine the next leg**. The cab stops to deliver and collect messages to/from a cluster head after each leg. Therefore, we do not need to plan the route beyond the next leg, as the future legs should be adaptively determined according to the messages the ferry collects in the future and have not been unveiled to the ferry yet. Therefore, if a scheme can produce each leg optimally, it offers a good solution to the MCR problem.

Based on the above observations, we use a weighting scheme to choose the "best"[2] cluster head among all the neighbors as the next waypoint for a cab to move to and this in turn will eventually forms an adaptive route. The Shortest Process Time First (SPTF) rule from the Job Scheduling Problem (JSP) is adopted to compute the weight of the clusters, and we refer to our proposed approach as the Adaptive Cab Route (ACR) algorithm.

5.2. The Shortest Process Time First (SPTF) Rule

In the Job Sequencing Problem (JSP) [20] of a single machine, we need to find the optimal sequence of a series of jobs k with weight[3] l_k and processing time p_k to minimize the average delay of these jobs which is given by:

$$\Delta_{jsp} = \frac{\Sigma_k l_k \delta_k}{\Sigma_k l_k}$$

Smith proved in [21] that the optimal solution to JSP can be produced by applying the Shortest Process Time First (SPTF) rule to sequence the jobs:

- **SPTF Rule**: Sequencing the jobs in order of non-decreasing ratio p_k / l_k produces an optimal schedule to minimize Δ_{jsp}.

Comparing Equation (6) with Equation (3), we can observe that the objective functions of JSP and MCR are similar (details will be discussed in the next Section). Hence SPTF would be a useful tool to construct adaptive ferry routes. In this paper, to ensure that the equations that we derive in the next Section are well defined, we

[2]As the most suitable waypoint in a cab route to minimize Δ_τ
[3]Larger l_k represents higher importance and vice versa.

define an inverse SPTF (iSPTF) rule:

- iSPTF Rule: To optimally sequence the jobs in JSP, the job with highest l_k / p_k value should be chosen first. It is easy to see that iSPTF is logically equivalent to SPTF. They are both able to solve JSP optimally.

5.3. The Adaptive Cab Route (ACR) Algorithm

We may interpret the event that the cab goes to a destination cluster *d* as job *d*. Assuming the cab is currently at cluster *s*, we could say that the required processing time of job *d* is the traveling time from cluster s to cluster d, *i.e.* ε_{sd}, which is similar to p_k in JSP.

In order to apply the iSPTF rule, we need to determine the weight l_k of these jobs. We note in JSP, the weight of a job indicates the how much each job contributes to the objective function Δ_{jsp}. Similarly, job d contributes to Δ_T in the MCR problem by delivering the messages with destinations in cluster d. Using $dest(i)$ to denote the index of the destination cluster of message i (*i.e.* message i is meant to be delivered to $C_{dest(i)}$), the total size of messages delivered by the cab in job d can be expressed as

$$\mathrm{M}_d = \sum_{i:dest(i)} \mu_i .$$

We use this value as the weight of job d (like l_k in JSP). Then we can choose the cluster with maximal ratio of Md/ε_{sd} as the next waypoint of the adaptive cab route according to the iSPTF rule.

We also note that MCR and JSP are still two different problems. Because in MCR, ε_{sd} varies when the cab's location (cluster *s*) changes, while in JSP p_k is constant. Therefore iSPTF may not provide the exact optimal solution to MCR. However, we believe it still provides a good indication of which cluster head should be chosen to be the next waypoint.

According to iSPTF, we could define a weighting function W_d as

$$W_d = \frac{\mathrm{M}_d}{\varepsilon_{sd}} \qquad (7)$$

A cluster with the maximal weight will be chosen as the next waypoint in the adaptive route. To break a tie, we define ϕ_d as the time of the most recent arrival of the cab at cluster *d*. Given any two clusters C_1 and C_2, if $\phi_1 < \phi_2$, it means that the time since the last cab's arrival to cluster C_1 is longer than that to cluster C_2. If the two clustersboth have the maximum weight W among all the clusters, C_1 will be chosen as the next waypoint. The algorithm shown in **Table 1** illustrates how the ACR algorithm determines the cab's next waypoint when it arrives cluster *s* at time Φ.

Table 1. ACR algorithm.

```
ACR(cluster s, time $\Phi$){
  initialize $W_d = 0$ and $M_d = 0$ for all $d$
  for each un-delivered message $i$ do                    %Loop 1%
     $\mathrm{M}_{dest(i)} += \mu_i$
  for each cluster $d$ do                                 %Loop 2%
     $W_d = \frac{\mathrm{M}_d}{\varepsilon_{sd}}$
  for all the clusters do                                 %Loop 3%
     find $d_{\max}$ of which $W_{d\max}$ is the maximum
  if there are multiple that $d$'s $W_{d'} = W_{d\max}$ then
     for all the $d$'s do                                 %Loop 4%
        find $d'_{\max}$ of which $\Phi d'_{\max}$ is the minimum
     the next waypoint is $C_{d'\max}$
else the next waypoint is $C_{d\max}$
set $\phi_s = \Phi$
}
```

Lemma 2. The complexity of the ACR algorithm is $O(n_c + n_m)$.

Proof: In Loop 1 of ACR, the algorithm goes through the cab's storage to calculate the accumulated size of messages that will be delivered to cluster $dest(i)$. Since the total number of messages is nm, Loop 1 takes $O(n_m)$ time in the worst case. To calculate the weight W_d for each cluster in Loop 2, the algorithm goes through all the n_c clusters and therefore $O(n_c)$ time is required. Similarly, in order to find the cluster with maximum weight, another $O(n_c)$ time is required in Loop 3. To break the tie, Loop 4 will be executed for $O(n_c)$ times in the worst case. The overall complexity for algorithm ACR is thus $O(n_c + n_m)$.

We note the complexity of the ACR algorithm (Lemma 2) as well as the DCD algorithm (Lemma 1) is much lower than most of the existing schemes with helping hosts. Moreover, being a fully localized scheme, MCab is extremely suitable for the MANETs where the resources are limited.

6. Simulation Results

To validate the effectiveness of the MCab scheme, extensive simulations are carried out. We randomly generate the network topology and traffic in C++ programs and compare the results with existing schemes.

We model a MANET constructed in a 500 m by 500 m area. The number of hosts in each cluster is set to 20 at the beginning of the simulations. The communication range of the hosts is 5 m, and the storage size is 10Mb. The hosts move at a speed of 10 m/s.

The traffic in the MANET can be specified by the number of messages per unit time, but as the size of messages also varies, it is more convenient to describe it as the size of data transmitted per unit time, *i.e.* kilobits per second (kb/s). We refer it as the volume of the traffic.

The number of pro-cabs may vary in the simulation. We define the effective number of pro-cabs as the average number of pro-cabs over time to describe how many pro-cabs are deployed in the MANET.

6.1. Performance of the DCD Algorithm

We start with our study on the performance of the DCD algorithm. The main objective of the DCD algorithm is to dynamically recruit cabs in the MANET to deliver inter-cluster messages.

The algorithm is controlled by two parameters, namely Ω (the upper bound of the weighted waiting delay) and Θ (the time duration that a host serves as a pro-cab). To deploy temp-cabs, the performance of the DCD algorithm is also closely related to the frequency in which the hosts move from one cluster to another. A variable ρ is defined as the average length of period (in seconds) that a host stays in a cluster before it moves to another one.

Figure 4 displays how the number of recruited cabs changes with the volume of inter-cluster traffic. The left vertical axis corresponds to the traffic volume while the number of pro-cabs are indicated on the right axis. To study the impact of Ω, Θ and ρ on the performance of the DCD algorithm, we choose two (high and low) values for each of these variables. Each of the figures corresponds to a particular combination of the values of Ω and ρ, and plots the two groups of results which correspond to the two values of Θ. As shown in **Table 2**, the values of Θ are 500 s and 100 s; the values of Θ are 2000 s and 200s; the values of ρ are 5000 s and 200s. The effective number of pro-cabs are also shown in **Table 2**.

1) The value of ρ: The value of ρ significantly changes the effective number of pro-cabs.

With a high value of ρ (ρ = 5000 s), the hosts tend to stay in the same cluster for a long time, and thus there are less chances that messages could be delivered by a temp-cab. Therefore, more pro-cabs have to be recruited to keep the waiting delay below Ω.

If the value of ρ is low (ρ = 200 s), it implies that the hosts frequently move among the clusters. Therefore a lot of inter-cluster messages could be delivered by temp-cabs, and there are fewer messages that are left on the cluster head. Therefore, fewer pro-cabs need to be recruited.

Table 2. Effective Number of Pro-Cabs.

	$\rho = 5000s$		$\rho = 200s$	
Ω	$\Theta = 200s$	$\Theta = 2000s$	$\Theta = 200s$	$\Theta = 2000s$
100s	2.81	3.78	1.13	1.35
500s	0.98	1.49	0.15	0.44

We can observe this phenomenon in **Figure 4(a)** and **Figure 4(c)** by comparing line 1 ($\Theta = 200\text{ s}$, $\Omega = 100\text{ s}$, $\rho = 5000\text{ s}$) with line 5 ($\Theta = 200\text{ s}$, $\Omega = 100\text{ s}$, ρ = 200 s). For the same values of Θ and Ω, more pro-cabs will be recruited when $\rho = 2000\text{ s}$, as depicted by line 1. When ρ decreases to 200 s, more temp-cabs could be used, and thus fewer pro-cabs are needed to deliver the messages, as shown by line 5. Similar fact can be observed by comparing line 2 with line 6, line 3 with line 7, and line 4 with line 8 in **Figure 4**. In **Table 2**, we can also see the effective number of pro-cabs is much smaller when ρ is low ($\rho = 200\text{ s}$).

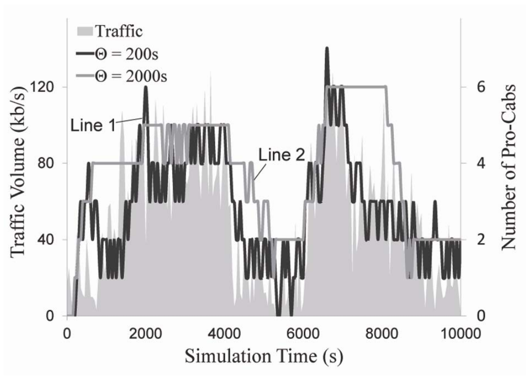
2) The value of Θ: Θ controls the duration for which a host serves as a pro-cab. It controls the frequency for triggering the pro-cab recruiting procedure, and thus affects how fast the system could respond to the change in traffic volume.

When the value of Θ is high ($\Theta = 2000\text{ s}$), the pro-cabs have a long service time and a slow retirement. Comparing line 1 with line 2 in **Figure 4(a)** on the time interval [7000, 8000], we can see that the number of cabs stays as high as 6 when $\Theta = 2000\text{ s}$ (as depicted by line 2) even though the traffic volume has already dropped to a lower level, where only about 3 pro-cabs will be deployed if $\Theta = 200\text{ s}$ (as depicted by line 1), meaning that several cabs may be unnecessary for the purpose of keeping the weighted waiting delay low. Similar phenomena can also be observed in the time interval [4000, 5000] in **Figure 4(a)**, [9000, 10000] in **Figure 4(b)**, and [8500, 10000] in **Figure 4(c)**, resulting in the effective numbers of cabs being much larger when Θ = 2000 s (as shown in **Table 2**).

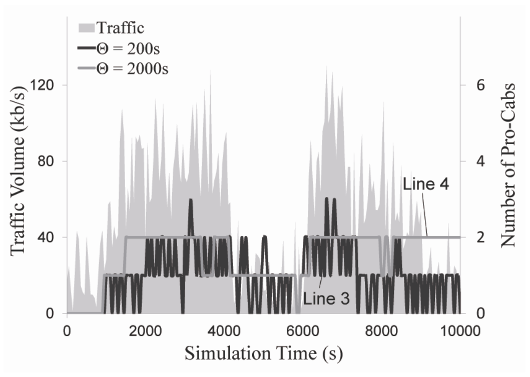
A low value of Θ ($\Theta = 200\text{ s}$) causes the pro-cabs to only serve for a short period of time, and change back to the state of normal host sooner. However, messages are still being generated and stored in the cluster heads, causing new pro-cabs to be recruited. The change in the cab number is thus more rapid and frequent than when $\Theta = 2000\text{ s}$. Since cabs are frequently recruited and retired and lines 1, 3, 5 and 7 appear to be more "spiky" than lines 2, 4, 6 and 8 respectively in **Figure 4**.

We should also note that frequent change in the number of cabs may also cause unwanted interruption to the other tasks of the hosts. For example, in order to collect reliable data, a sensor may need to stay stationary at the same position for some minimum duration. Hence low values of Θ may affect the efficiency of the network task, although it is able to help to reduce the number of pro-cabs.

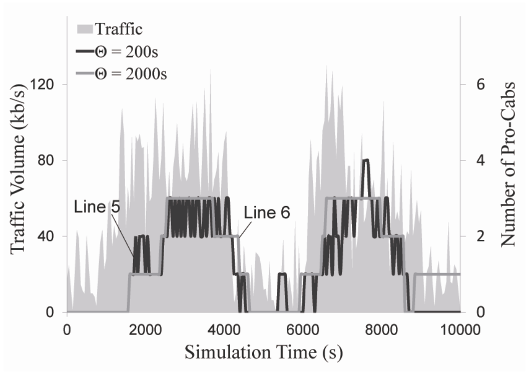
3) The value of Ω: Comparing **Figure 4(a)** with **Figure 4(b)**, or **Figure 4(c)** with **Figure 4(d)**, we can also observe the fact that the value of Ω influences the number of cabs more directly.



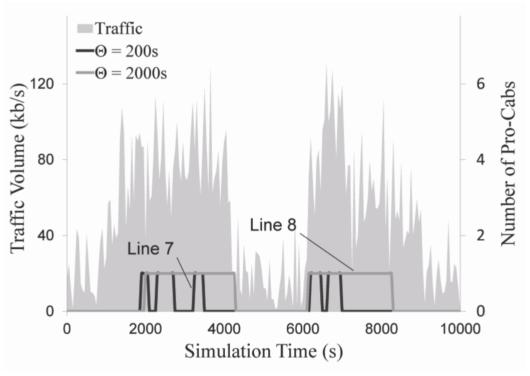
(a) Ω = 100 s, ρ = 5000 s



(b) Ω = 500 s, ρ = 5000 s



(c) Ω = 100 s, ρ = 200 s



(d) Ω = 500 s, ρ = 200 s

Figure 4. Performance of the DCD algorithm.

When the value of is high ($\Omega = 500\text{ s}$), the cluster heads are able to tolerate larger weighted waiting delay before recruiting new pro-cabs, and less pro-cabs will be hired in the MANETs. This is the reason why in **Table 2**, the effective numbers of pro-cabs are much smaller when $\Omega = 500\text{ s}$.

On the other hand, if the value Ω is low ($\Omega = 100\text{ s}$), the cluster heads have to frequently hire new pro-cabs to keep the waiting delay of the messages low. Comparing **Figure 4(a)** with **Figure 4(b)**, or **Figure 4(c)** with **Figure 4(d)**, it is easy to observe that the number of pro-cabs significantly increases when the value of Ω is low.

In general, our simulation results show that the DCD algorithm can effectively adapts the number of cabs to the traffic volume of the network. We will also show how it bounds the weighted waiting delay below Ω in Section VI-C. Before that, the performance of the ACR algorithm will be discussed.

6.2. Performance of the ACR Algorithm

To evaluate the performance of the ACR algorithm, we assume that cabs are selected beforehand and are randomly deployed without the use of the DCD algorithm in this section. We compare the delay incurred by the adaptive route constructed by the ACR algorithm with the most commonly adopted solution, *i.e.* the TSP route. It is well-known that TSP is a NP-hard problem, where optimal solution cannot be found within polynomial time. We use brute force to find the optimal TSP route in the simulation. However it would only be feasible to use To address this issue, we also use the Improving Search Algorithm [22] to find an approximate solution to TSP as a feasible route. Both optimal and approximate TSP routes are static and are used as benchmarks to compare with the routes obtained by the ACR algorithm.

10 clusters of hosts are considered for this simulation. The cabs are randomly allocated in different clusters initially, and move according to the routes generated by the ACR algorithm, optimal TSP algorithm, or approximate TSP algorithm. The results are shown in **Figure 5**.

We can see that increasing the number of cabs reduces the average delay of the ACR routes, but no significant change can be observed for TSP and approximate TSP routes. This is because when the same static route is used, the delay of a message only depends on the distance between its source and destination on the route, and is not affected by the number of cabs. Therefore once the route is constructed, the delay of messages will not be affected by the change in the number of cabs.

On the other hand, when the ACR algorithm is used, each cab defines its own route based on the messages it

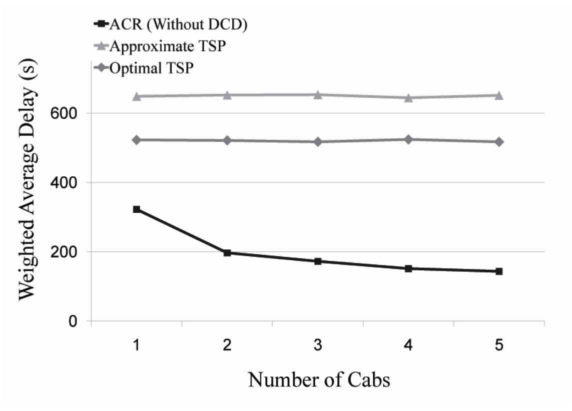


Figure 5 . Performance of the ACR algorithm (traveling delay).

is carrying. The delay of a message can be further shortened if there are fewer messages to be carried by a single cab. For example, if a cab only carries one message, it will directly move to the destination cluster of the message. A fair amount of time can be thus saved. As the number of cabs increases, fewer messages will be carried by each of them, and the average delay is thus further decreased. Moreover, since the ACR algorithm is fully distributed, no collaboration is needed among the cabs. Each of them can decide its route locally. The increase in cab number does not affect the complexity of the scheme at all.

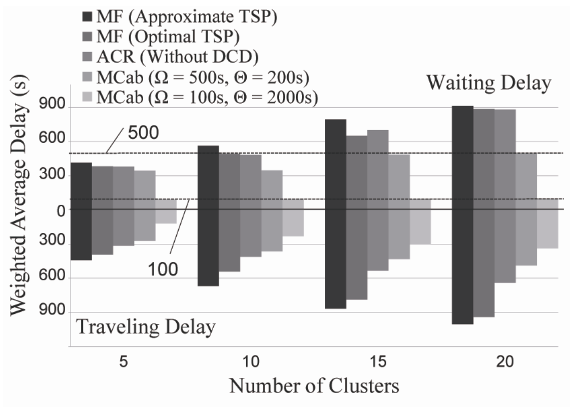
6.3. Overall Performance of the MCab Scheme

In the MCab scheme, when both the DCD and ACR algorithms are applied for the cab deployment and route design, the average weighted delay is further reduced. We demonstrate this result in **Figure 6** to **Figure 8** by showing how the MCab scheme perform when there are different number of clusters (n_c). The three figures displays the weighted average delay of the messages, the effective number of pro-cabs and the percentage (in size) of messages delivered by procabs, respectively. In each figure, two values of ρ (5000 s and 200 s) are used in the simulation. Two different implementations of the DCD algorithm ($\Omega = 100\text{ s}$, $\Theta = 2000\text{ s}$ and $\Omega = 500\text{ s}$, $\Theta = 200\text{ s}$) are demonstrated. The results for the other two cases ($\Omega = 500\text{ s}$, $\Theta = 2000\text{ s}$ and $\Omega = 100\text{ s}$, $\Theta = 200\text{ s}$) exhibit similar results, and are thus omitted in this section

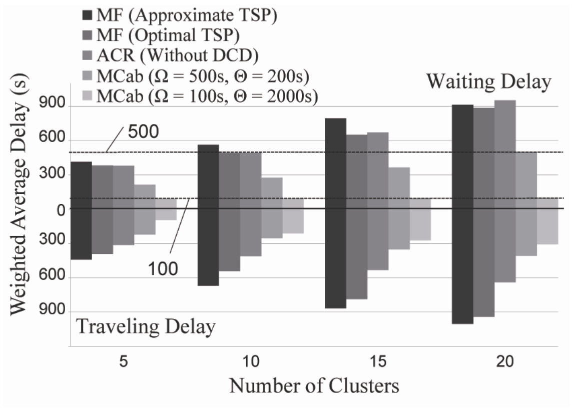
In **Figure 6** the simulation results are shown as a group of columns. The upper part (above 0) of each column shows the duration of average waiting delay Δ_ω, and the lower part (below 0) shows the duration of average traveling delay Δ_T. As a result, the entire bar length shows the overall average delay Δ of the corresponding scheme. We compare the results with the case where a single helping host is deployed in theMF scheme, and moves on a TSP route or an approximate TSP route. Moreover, we also plot the values of the average weighted delay when the ACR algorithm is applied to a single cab without the DCD algorithm in the figure as a reference.

It can be seen from **Figure 6** that the total delay is much lower with the MCab scheme. Let's consider the case when there are 5 clusters. We can observe that for this case the MF scheme with approximate TSP incurs the longest message delay. It is followed by the MF scheme with optimal TSP, because the optimal route of TSP will be shorter than an approximate route. As we have shown in Section VI-B, the ACR algorithm will reduce the traveling delay by using adaptive cab routes instead of fixed ones. However, as the ACR algorithm calculates the weights of the clusters based on the messages that have been stored in the cab, it does not guarantee a lower waiting delay for those messages that have not yet been uploaded to the cabs. Therefore, the waiting delay is not significantly reduced by the ACR algorithm alone (without the DCD algorithm). Moreover, it can be observed in some particular incidences that it may incur a slightly higher waiting delay than the MF schemes (e.g. $\rho = 5000\text{ s}$ with 15 clusters). When the DCD algorithm is also applied (i.e. the MCab scheme), the waiting delay is further reduced, because some of the messages can be carried by the temp-cabs and do not need to wait for the pro-cabs. The message delay reduces to the minimum when the MCR scheme with $\Omega = 100\text{ s}$ and $\Theta = 2000\text{ s}$ is adopted. This is because when $\Omega = 100\text{ s}$, more pro-cabs are deployed than the case when $\Omega = 500\text{ s}$ as we discussed in Section VI-A. The same fact can also be observed from **Figure 7**, where on average 4.5 pro-cabs are deployed to serve 5 clusters when $\rho = 5000\text{ s}$, $\Omega = 100\text{ s}$ as compare to 1.3 pro-cabs when $\Omega = 500\text{ s}$. We note the other cluster sizes in **Figure 6** also exhibit the same performance trend described above.

It should also be pointed out that when there are 5 clusters and $\rho = 5000\text{ s}$, the difference between the results for MCab ($\Omega = 500\text{ s}$, $\Theta = 2000\text{ s}$) and ACR (without DCD) is not as significant as when there are more clusters, or when the value of ρ is lower ($\rho = 200\text{ s}$, in **Figure 6(b)**). This is because the waiting delay in this case is lower than 500 s even when there is only one cab. As a result, we note that the average number of cabs is also about 1 when the DCD algorithm is used, which means it is not necessary to deploy more cabs to further reduce the average weighted waiting delay in the MCab scheme. Moreover, since the number of temp-cabs deployed in this scenario is also very few as ρ is large, the performance is very similar to the ACR (without DCD) case.

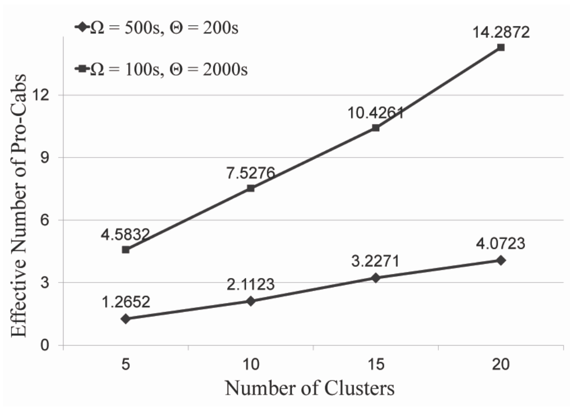


(a) = ρ 5000 s

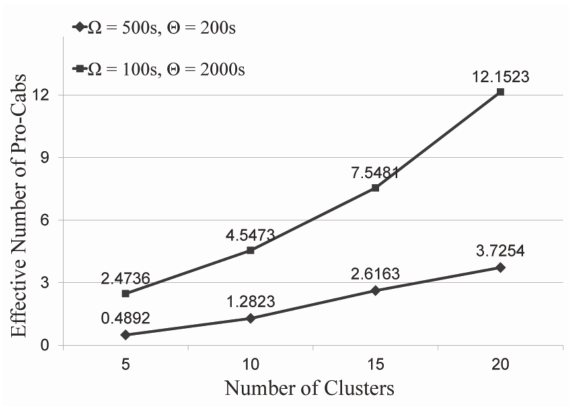


(b) = ρ 200 s

Figure 6. Delay of messages.

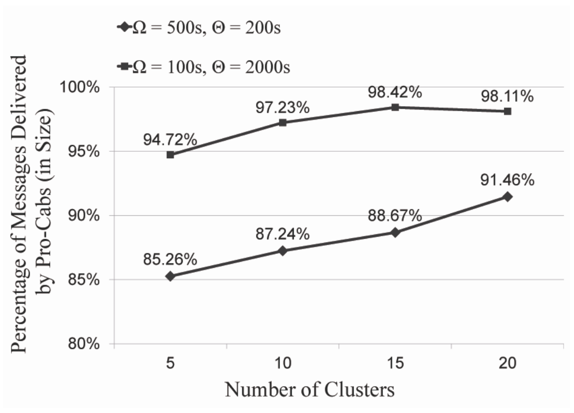


(a) ρ = 5000 s

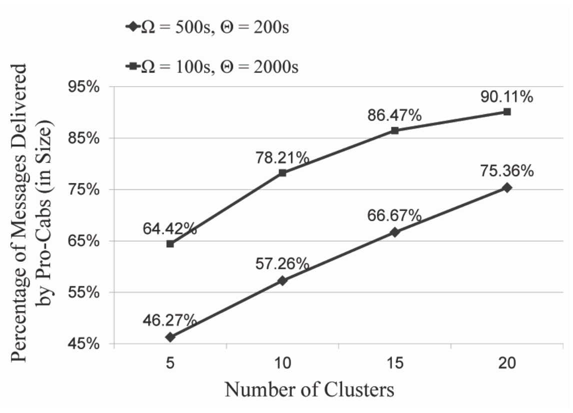


(b) ρ = 200 s

Figure 7. Effective number of pro-cabs.



(a) ρ = 5000 s



(b) ρ = 200 s

Figure 8. Percentage of messages delivered by pro-cabs.

When the number of clusters increases, the weighted average waiting delay with the ACR algorithm (without DCD) increases as well. This is due to the fact that the pro-cab needs to visit more clusters before delivering the message to its destination. As a result, when the DCD algorithm is applied (*i.e.* the MCab scheme), more pro-cabs will be recruited to lower the waiting delay so that it does not exceeds the predetermined upper bound Ω. Moreover, we can observe that under the control of the DCD algorithm, the weighted average waiting delay of the messages are bounded by the values of Ω, which are indicated by the respective reference lines (500 s and 100 s) in **Figure 6**.

The effectiveness of deploying temp-cabs can also be demonstrated by comparing the height of the columns representing MCab scheme ($\Omega = 500$ s, $\Theta = 200$ s) in **Figure 6(a)** and **Figure 6(b)**, where the values of weighted average waiting delay decrease with the value of ρ. As ρ decreases from 5000 s to 200 s, more temp-cabs can be used for inter-cluster message delivery, thus less pro- cabs are recruited (as depicted in **Figure 7**), and a lower percentage of messages will be delivered by the pro-cabs (as depicted in **Figure 8**). It thus reduces the waiting delay of the messages, since some of them are taken to their respective destination clusters by temp-cabs before the timer expires. This phenomenon cannot be observed when $\Omega = 100$ s, $\Theta = 2000$ s, and when there are 20 clusters for $\Omega = 500$ s, $\Theta = 200$ s, because in these cases additional pro-cabs are recruited in order to bound the waiting delay below Ω.

Interestingly, the traveling delay is also slightly reduced by using more temp-cabs. This is because the messages carried by a temp-cab are selected based on their destinations. The temp-cab takes them from the source cluster head and moves directly to their destination cluster, while a pro-cab is shared by messages with different destinations, and may have to visit several other clusters (as destinations of other messages stored in the pro-cab) before eventually deliver these messages. However, as we can see from **Figure 8**, the majority of messages are delivered by the procabs in most of the scenarios, and the weighted average traveling delay is still dominated by the routes of pro-cabs, *i.e.* the ACR algorithm.

In conclusion, simulation validates that both the DCD and ACR algorithms works as we expected, and the MCab scheme effectively reduces the average weighted waiting delay.

7. Conclusions

In this paper, we propose a scheme with flexible helping hosts, namely Message Cab (MCab) for message delivery in partitioned MANETs. It consists of two algorithms, referred to as the Dynamic Cab Deployment (DCD) algorithm and the Adaptive Cab Route (ACR) algorithm. The DCD algorithm dynamically selects hosts to become the helping hosts (cabs) and move among the clusters to deliver inter-cluster messages, and the ACR algorithm designs the route of the cabs so that the delay of the messages can be reduced. Comparing with the existing schemes with helping hosts, we have shown that the MCab scheme effectively shorten the delay of the messages with the simulations and is adaptive to the varying traffic in the network, which has not been discussed in the existing works.

8. References

[1] S. Corson and J. Macker, "Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations," United States, 1999.

[2] W. Zhao and M. Ammar, "Message Ferrying: Proactive Routing in Highly-Partitioned Wireless Ad Hoc Networks," *Proceedings of the* 9*th IEEE Workshop on Future Trends of Distributed Computing Systems* (*FTDCS'03*), Washington, DC, 2003, pp. 308-314.

[3] W. Zhao, M. Ammar and E. Zegura, "A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad hoc Networks," *Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing* (*MobiHoc'*04), New York, 2004, pp. 187-198.

[4] W. Zhao, M. Ammar and E. Zegura, "Controlling the Mobility of Multiple Data Transport Ferries in a Delay-Tolerant Network," *Proceedings of the* 24*th Annual Joint Conference of the IEEE Computer and Communications Societies* (*INFOCOM'*05), Miami, Vol. 2, 2005, pp. 1407-1418.

[5] M. M. B. Tariq, M. Ammar and E. Zegura, "Message Ferry Route Design for Sparse Ad Hoc Networks with Mobile Nodes," *Proceedings of the* 7*th ACM international symposium on Mobile Ad Hoc Networking and Computing* (*MobiHoc'*06), New York, NY, 2006, pp. 37-48.

[6] M. Ye, X. Tang and D. L. Lee, "Fair Delay Tolerant Mobile Data Ferrying," *Proceedings of the 10th International Conference on Mobile Data Management: Systems, Services and Middle-ware* (*MDM'*09). Washington, DC, 2009, pp. 182-191.

[7] M. H. Ammar, D. Chakrabarty, A. D. Sarma, S. Kalyanasundaram and R. J. Lipton, "Algorithms for Message Ferrying on Mobile Ad Hoc Networks," *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science* (*FSTTCS'*09), IIT Kanpur, 2009, pp. 13-24.

[8] Q. Li and D. Rus, "Communication in Disconnected Ad Hoc Networks Using Message Relay," *Journal of Parallel and Distributed Computing*, Vol. 63, No. 1, 2003, pp. 75-86.

[9] S.-H. Chung, K.-F. Ssu, C.-H. Chou, and H. C. Jiau, "Improving Data Transmission with Helping Nodes for Geographical Ad Hoc Routing," Computer Networks, Vol. 51, 2007, pp. 4997-5010.

[10] C.-H. Ou, K.-F. Ssu and H. C. Jiau, "Connecting Network Partitions with Location-Assisted Forwarding Nodes in Mobile Ad Hoc Environments," *Proceedings of the* 10*th IEEE Pacific Rim International Symposium on Dependable Computing* (*PRDC'*04), Washington, DC, 2004, pp. 239-247.

[11] D. Borsetti, C. Casetti, C.-F. Chiasserini, M. Fiore and J. M. Barcel'o-Ordinas, "Virtual Data Mules for Data Col-

lection in Road-Side Sensor Networks," *Proceedings of the 2nd International Workshop on Mobile Opportunistic Networking* (*MobiOpp'*10). New York, 2010, pp. 32-40.

[12] R. C. Shah, S. Roy, S. Jain and W. Brunett, "Data MULEs: Modeling a Three-tier Architecture for Sparse Sensor Networks," Technical Report IRS-TR-03-001, Intel Research Lab at Seattle , 2003.

[13] D. Jea, A. Somasundara and M. Srivastava, "Multiple Controlled Mobile Elements (Data Mules) for Data Collection in Sensor Networks," *Lecture Notes in Computer Science*, Vol. 3560, No. 2005, 2005, pp. 244-257.

[14] A. A. Hasson, R. Fletcher and A. Pentland, "DakNet: A Road to Universal Broadband Connectivity," First Mile Solutions, Technical Report, 2003.

[15] I. Rhee, M. Shin, S. Hong, K. Lee and S. Chong, "On the Levy-Walk Nature of Human Mobility: Do Humans Walk like Monkeys?" *Proceedings of the* 27*th Annual Joint Conference of the IEEE Computer and Communications Societies* (*INFOCOM* '08), Rio de Janeiro, 2008.

[16] Y. Zhang, C. P. Low, J. M. Ng and T. Wang, "An Efficient Group Partition Prediction Scheme for MANETs," *Proceedings of the* 2009 *IEEE Wireless Communications and Networking Conference* (*WCNC'*09), Budapest, 2009, pp. 1-6.

[17] R. Sitters, "The Minimum Latency Problem is NP-Hard for Weighted Trees," In: W. Cook and A. Schulz, Ed., *Integer Programming and Combinatorial Optimization, ser. Lecture Notes in Computer Science*, Springer Berlin/ Heidelberg, Vol. 2337, 2006, pp. 230-239.

[18] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan and M. Sudan, "The Minimum Latency Problem," *Proceedings of the* 26*th annual ACM symposium on Theory of computing* (*STOC'*94), New York, NY, USA: ACM, 1994, pp. 163-171.

[19] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, "Proof Verification and the Hardness of Approximation Problems," *Journal of the ACM*, Vol. 45, No. 3, 1998, pp. 501-555.

[20] K. R. Baker, "Introduction to Sequencing and Scheduling," John Wiley & Sons, Inc, New York, USA, 1974.

[21] W. E. Smith, "Various Optimizers for Single-Stage Production," *Naval Research Logistics Quarterly*, Vol. 3, No. 1-2, 1956, pp. 59-66.

[22] P. Merz and B. Freisleben, "Genetic Local Search for the TSP: New Results," *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, Indianapolis, USA, April 1997, pp.159-164.